Université de Lille

TP 5.03 : MAVEN

R5.A.09 - Frameworks WEB
BUT3 - Info
Philippe Mathieu
2024–2025

Objectifs

Savoir utiliser Maven. Gérer des applications Java et des applications Java EE. Améliorer le `pom.xml`.

Exercice 1 : Une simple application Java (1h)

Q1. Créez en tapant simplement `mvn archetype:generate` un projet java classique à partir de l'archétype `maven-archetype-quickstart` (a priori num 2186)(prendre la version 1.0) avec comme groupID `fr.but3` et comme nom de projet `tp503a`

Q2. Modifiez le `pom` généré pour ajouter les propriétés de compilation et d'encodage.

Q3. Par défaut, Maven crée un fichier `App.java` qui contient un `"Hello world"` qui nous suffira pour cet exercice.

Q3.1. Testez les cibles classiques : `compile`, `clean`, `package` plusieurs fois en regardant à chaque fois le résultat obtenu à l'aide de la commande unix `tree`.

Q3.2. Testez trois méthodes d'exécution :

1. Par la commande `java` et lancement de la classe `App`
   .....
2. Par la commande `java` en s'appuyant sur le jar obtenu
   .....
3. Par la commande `mvn` et lancement de la classe `App`.
   Cette dernière commande est toujours préférable puisqu'elle prend automatiquement en compte toutes les dépendances. Les 2 méthodes précédentes nécessitent de citer tous les jars dans le `-cp`)
   .....

Q3.3. Ajoutez la propriété `exec.mainClass` dans `properties` pour faciliter le lancement ultérieur. Testez sa bonne prise en compte.

Q4. Intégrez ce projet dans votre IDE préféré (VSCode : `File/Open`, une perspective Maven s'ouvre automatiquement ; Eclipse : `import/Existing maven project`).

Q5. Créez dans l'IDE un nouveau but de lancement à partir de `Run As` sur le nom de projet, puis `Maven build ...` puis Goal : clean install.

Q6. En modifiant le `println` du programme, vérifiez que vous savez compiler, exécuter, modifier ce projet aussi bien sous votre IDE qu'en utilisant Maven en ligne de commande.

Exercice 2 : Gestion des dépendances et des plugins (1/2h)

Q1. En vous aidant du site MVN Repository, ajoutez une dépendance au driver `postgresql` et ajoutez au programme principal une simple connexion/fermeture à la base de données.

Q2. Exécutez ce projet.

Q3. Affichez la liste des dépendances du projet.

Q4. Testez l'exportation des dépendances dans un répertoire de votre choix (`/tmp` par exemple)

Q5. Lancez maintenant votre application via la commande `java` et en utilisant directement le `.class` obtenu.

Q6 et Q7 uniquement si vous êtes dans les temps. Sinon, sautez.

Q6. Le `maven-jar-plugin` par défaut n'est pas paramétré pour faire un jar exécutable. Pour réaliser cela il faut surcharger le `maven-jar-plugin` avec les bons paramètres pour constituer un `MANIFEST.MF` correct. Le `addClasspath` à `True` indique à Maven d'y ajouter un classpath indiquant où sont les jars. Vérifiez la présence de ce fichier et le classpath indiqué dedans !

Q7. Le fichier jar est destiné à être distribué. Copiez ce `jar` dans votre répertoire `tmp`, et faites en sorte de l'exécuter.

Q8. Le répertoire `.m2` contient tout ce que Maven télécharge pour vous. Il devient vite très volumineux. Faites un `tree` dans ce répertoire et un `du -k`. Histoire de tout remettre au propre, détruisez votre répertoire `.m2` et relancez un `mvn clean package` pour le reconstruire.

Exercice 3 : Accès à une ressource (1/2h)

Q1. Créez un fichier `Properties` nommé `data.txt` avec une ligne `clé=valeur` quelconque dans `main/resourc` (par ex nom=dupont)

Q2. Accédez à cette ressource dans le programme principal (via le `ClassLoader`) et affichez son contenu.

Q3. Vérifiez par un `tree` l'endroit où est placée cette ressource.
Constatez que votre fichier ressources a été recopié dans la zone d'exécution, d'où l'importance du classloader pour le retrouver.

Exercice 4 : Une application WEB (1h)

Q1. Créez un projet web Java EE à partir de l'archétype `maven-archetype-webapp` (a priori num 2191) avec un groupID `fr.but3` et comme nom de projet `tp503b`

Q2. Mettez à jour les propriétés (notamment version du compilateur et encodage).

Q3. Compilez ce projet avec `mvn package`. Regardez l'arborescence obtenue. Vous constatez que Maven a créé un répertoire `target/tp503b` qui est une vraie structure JEE. Il n'y a donc plus qu'à l'intégrer dans un serveur JEE pour qu'elle fonctionne.

Q4. Testez cette première page dans votre Tomcat
Pour cela, Tomcat arrêté, créez un lien symbolique dans votre `tomcat/webapps` pointant sur `target/tp503b` puis `http://localhost:8080/tp503b/`

Q5. Arrêtez votre Tomcat, et lancez maintenant ce contexte avec le plugin `cargo` de chez Codehaus sans rien déclarer dans le pom. Quand rien n'est déclaré dans le pom, le lancement d'un plugin se fait via `mvn groupID:artifactId:but`, donc ici :

```
mvn org.codehaus.cargo:cargo-maven3-plugin:run
```

Q6. Si vous déclarez le plugin dans le `pom`, il est alors possible de taper plus simplement `mvn cargo:run`

```
<plugin>
    <groupId>org.codehaus.cargo</groupId>
    <artifactId>cargo-maven3-plugin</artifactId>
    <version>1.10.14</version>
</plugin>
```

Q7. Ce plugin lance par défaut un serveur `jetty`. Si vous souhaitez un serveur tomcat, il faut rajouter la propriété `<cargo.maven.containerId>tomcat10x</cargo.maven.containerId>`

Q8. Ajoutez les dépendances `Postgres` et `jakarta.servlet-api`

Q9. Affichez le `effective-pom`. Quelles sont les versions du `maven-compiler-plugin` et du `maven-war-plugi` qui sont utilisées avec votre pom ?

Q10. Après avoir effectué un `mvn package`, listez le contenu du war obtenu. Vous constatez que `postgres.jar` est dans le répertoire `lib` tandis que `servlet-api` n'y est pas. pourquoi ?

Q11. Recopiez la servlet `Lister` qui liste la table `etudiant`, issue du précédent TP, dans `src/main/java/fr/but` Effectuez un `mvn clean package`

Q12. Vérifiez toujours votre arborescence ! notamment `target/tp503b`. Il doit avoir la bonne structure JEE ! (vérifiez notamment que les `.class` sont bien dans `WEB-INF/classes`

Q13. Testez cette servlet via votre serveur tomcat externe, et via le plugin `cargo` fourni par Maven

Q14. Intégrez le projet à votre IDE, testez une modif de la servlet et créez dans votre IDE une configuration `run as` adaptée pour test.

Q15. Il ne vous reste plus ... qu'à y mettre du style, avec la feuille du `TP501` par exemple. Les feuilles de style se placent au même endroit que les JSP, dans `src/webapp`.

Conclusion

A la fin de ce TP vous avec maintenant deux exemples de structures de base d'un projet Java et d'un projet Java EE. Conservez les ;-)

Dorénavant, soit vous continuez avec votre serveur Tomcat et des liens symboliques, soit vous continuez avec Cargo